**Documentation for Kracht Security Cost Analyzer**

**Application Overview**

**Name**: Kracht Security Cost Analyzer  
**Purpose**: This application is a web-based tool designed to help enterprises monitor and manage their Azure security costs. It provides a centralized dashboard to visualize total costs, security component breakdowns (e.g., Defender, Firewall, WAF), subscription costs, and top resource usage, enabling users to control and optimize their cloud security expenses.

**Target Users**: IT administrators, financial analysts, and Azure cloud managers within enterprises.

**Key Features**:

* **Cost Overview**: Displays total cost, security cost, and non-security cost breakdowns.
* **Security Components**: Detailed costs for Defender, DDoS Protection, Key Vault, Firewall, WAF, and Sentinel.
* **Subscription Analysis**: Breakdown of costs by subscription.
* **Top Resources**: Identifies the top 5 Defender resources by cost.
* **Export Functionality**: Allows exporting cost data to CSV.
* **Granularity**: Supports daily, weekly, or monthly cost views (if backend supports it).
* **Authentication**: Secured with Microsoft Azure Active Directory (AAD) using MSAL.

**How It Works**

* **Frontend**: Built with React, the application runs on http://localhost:8080 (configurable via PORT environment variable). It uses MSAL for authentication, rendering components like Login.js, Dashboard.js, OverviewSection.js, and DefenderCostSection.js.
* **Backend**: A Node.js server runs on http://localhost:3001 (configurable), handling API requests (e.g., /totalCostRouter, /defenderCost) and authenticating with Azure AD tokens. It fetches cost data from the Azure Cost Management API and serves it to the frontend.
* **Data Flow**:
  1. User logs in via Microsoft credentials, triggering MSAL to obtain an access token.
  2. The frontend uses the token to request data from backend endpoints.
  3. The backend validates the token, queries Azure APIs, and returns JSON data (e.g., total cost, security components).
  4. The frontend renders charts and tables using Chart.js and DataTable components.
* **Authentication**: MSAL handles the OAuth 2.0 flow, with the backend using JWT validation to secure API endpoints.

**Step-by-Step Guide to Run the Application**

**Prerequisites**

* **Node.js**: Version 14.x or higher (check with node -v).
* **npm**: Included with Node.js (check with npm -v).
* **Git**: Optional, for version control (check with git --version).
* **Windows**: Ensure CMD or PowerShell is available.
* **Azure AD Credentials**: Client ID, tenant ID, and permissions for Azure Cost Management API.

**Steps**

1. **Navigate to Project Directories**:
   * Open two Command Prompt windows.
   * In the first window:

cmd

cd C:\azurecostapp-local

* + In the second window:

cmd

cd C:\azurecostapp-frontend-react

1. **Install Dependencies**:
   * For the backend:

cmd

npm install

* + For the frontend:

cmd

npm install

1. **Start the Backend Server**:
   * In the first window:

cmd

node server.js

* + Expected output: Server running at http://localhost:3001.
  + Test: Open http://localhost:3001/ in a browser (should show "Backend Server is Running").

1. **Start the Frontend Server**:
   * In the second window:

cmd

set PORT=8080 && npm start

* + Expected output: Compiled successfully! with Local: http://localhost:8080.
  + Test: Open http://localhost:8080 in a browser (should show the login page).

1. **Log In**:
   * Click "Sign In with Microsoft" on the login page.
   * Enter your Azure AD credentials and complete MFA.
   * After authentication, you should be redirected to the dashboard.
2. **Verify Functionality**:
   * Check the Overview section for Total Cost (should be 141.48 based on your data).
   * Navigate to other sections (e.g., Defender Cost) and verify data loading.
3. **Stop the Servers**:
   * Press Ctrl+C in each terminal to stop the servers when done.

**Notes**

* If port 8080 is in use, change PORT to another value (e.g., 8081) or free the port using netstat -aon | findstr :8080 and Task Manager.
* Ensure .env files (if used) are configured with Azure credentials.

**Important Details to Share with AI**

To ensure the AI (like me) can assist you efficiently next time, provide the following details when starting a new conversation or encountering issues:

* **Project Structure**:
  + Backend directory: C:\azurecostapp-local
  + Frontend directory: C:\azurecostapp-frontend-react
  + Key files: server.js, routes/totalCostRouter.js, auth.js, index.js, Login.js, Dashboard.js, OverviewSection.js, useDashboardData.js
* **Ports**:
  + Backend: 3001 (default), configurable to 8080 or others.
  + Frontend: 8080 (via set PORT=8080 && npm start), default 3000.
* **Authentication**:
  + MSAL with Azure AD.
  + Client ID: 64fac289-ad7f-44e4-a719-a24dc9ced789
  + Tenant ID: e7ea7b4d-a56f-426b-be45-857324a57837
  + Redirect URI: http://localhost:8080 (update if changed).
  + Scopes: https://management.azure.com/user\_impersonation
* **Backend Endpoints**:
  + /totalCostRouter, /subscriptions, /defenderCost, etc.
  + Expected response structure (e.g., totalCost, securityComponents).
* **Errors Encountered**:
  + Include terminal output, browser console logs (F12), and network tab details.
* **Previous Issues**:
  + Mention past problems (e.g., 429 Too Many Requests, "Cannot GET /", "localhost refused to connect") and their resolutions.
* **Custom Configurations**:
  + Any .env variables, custom scripts, or port redirections (e.g., netsh rules).

**Additional Notes**

* **Documentation Maintenance**: Save this document in your project folder (e.g., C:\azurecostapp-frontend-react\docs\README.md) for future reference.
* **Backup**: Regularly back up your project directories to avoid losing progress.
* **Updates**: Check for Node.js, npm, and react-scripts updates periodically (npm outdated).

This documentation should make it easy to run your application next time and provide the AI with the context needed for effective support. Let me know if you need further refinements or help with specific sections!